Benchmarking Adaptive Indexing

Goetz Graefe, Stratos Idreos, Harumi Kuno, *Stefan Manegold*

HP Labs
Palo Alto, CA
first.last@hp.com
www.hpl.hp.com

CWI Amsterdam
The Netherlands
first.last@cwi.nl
www.cwi.nl

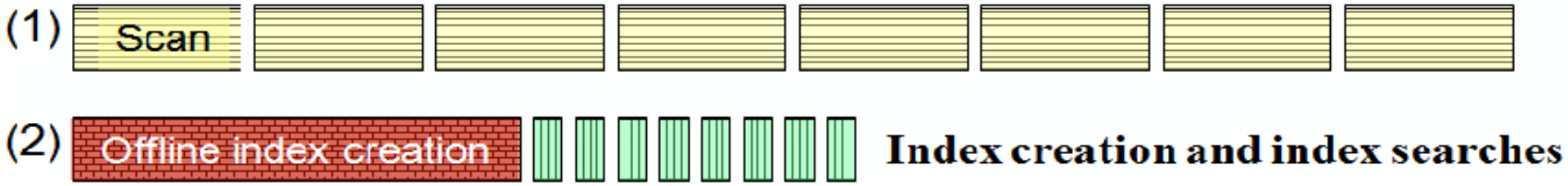
TPCTC 2010
Singapore

Adaptive Indexing???

Index Creation vs. Query Processing (1/4)

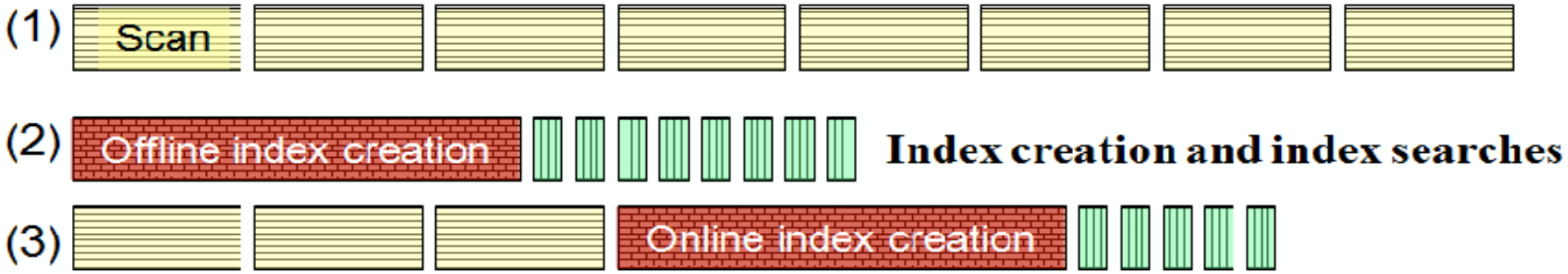
+ No investment
+ No storage cost
+ No maintenance overhead
- Base-line performance

Index Creation vs. Query Processing (2/4)



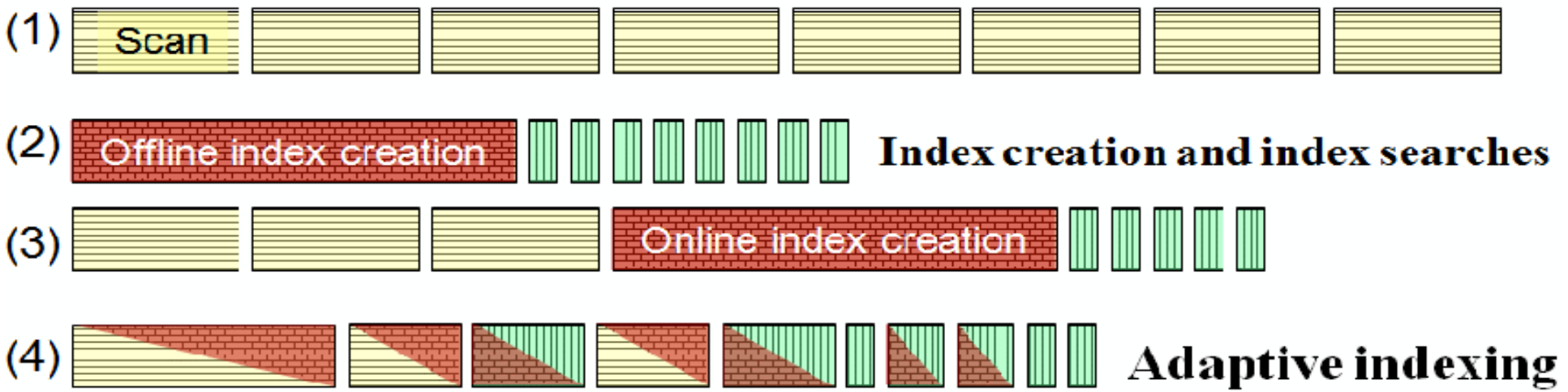
Index Creation vs. Query Processing (2/4)

Index Creation vs. Query Processing (3/4)



Index Creation vs. Query Processing (3/4)

Index Creation vs. Query Processing (4/4)



Index Creation vs. Query Processing (4/4)

Database Cracking

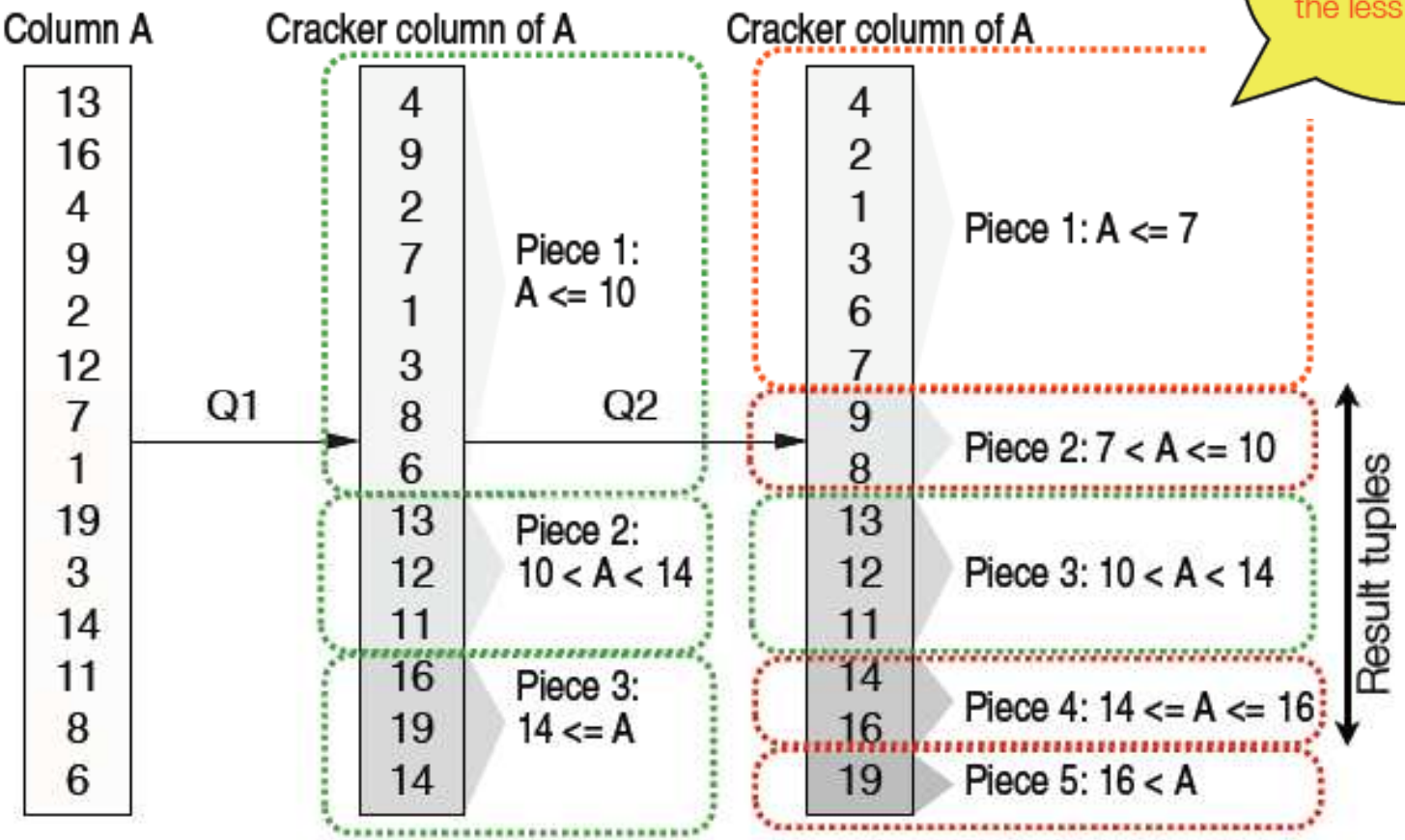
Each query is treated as an advice on how data should be stored

Physically reorganize based on the selection predicate

The more we crack, the more we learn, the less we touch

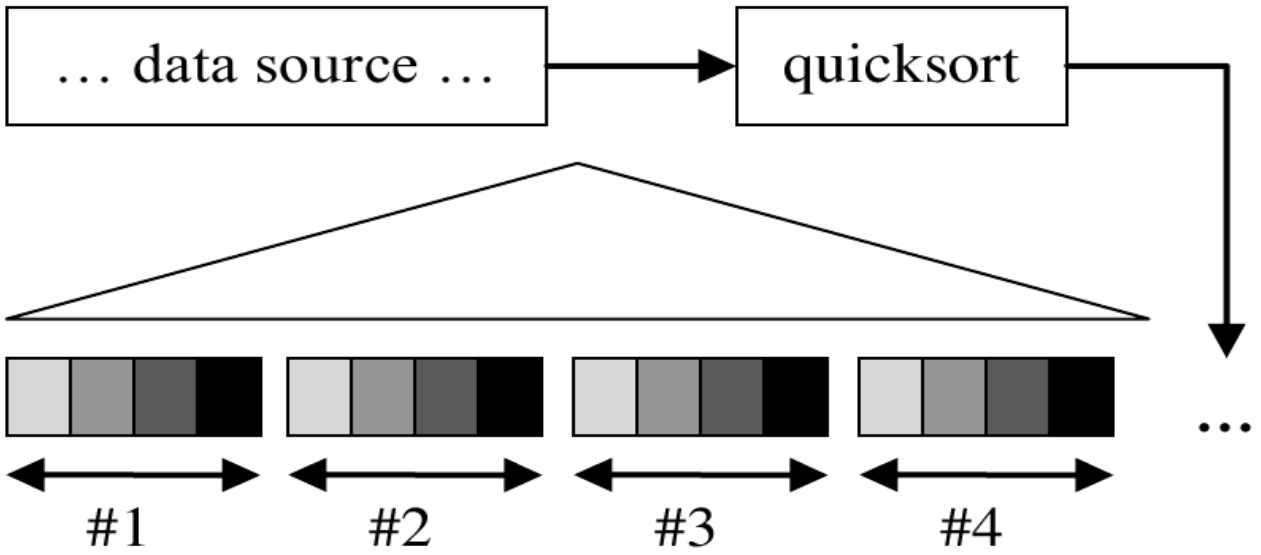
Q1:
select *
from R
where R.A > 10
and R.A < 14

Q2:
select *
from R
where R.A > 7
and R.A <= 16



"Incremental Quick-Sort"

Adaptive Merging

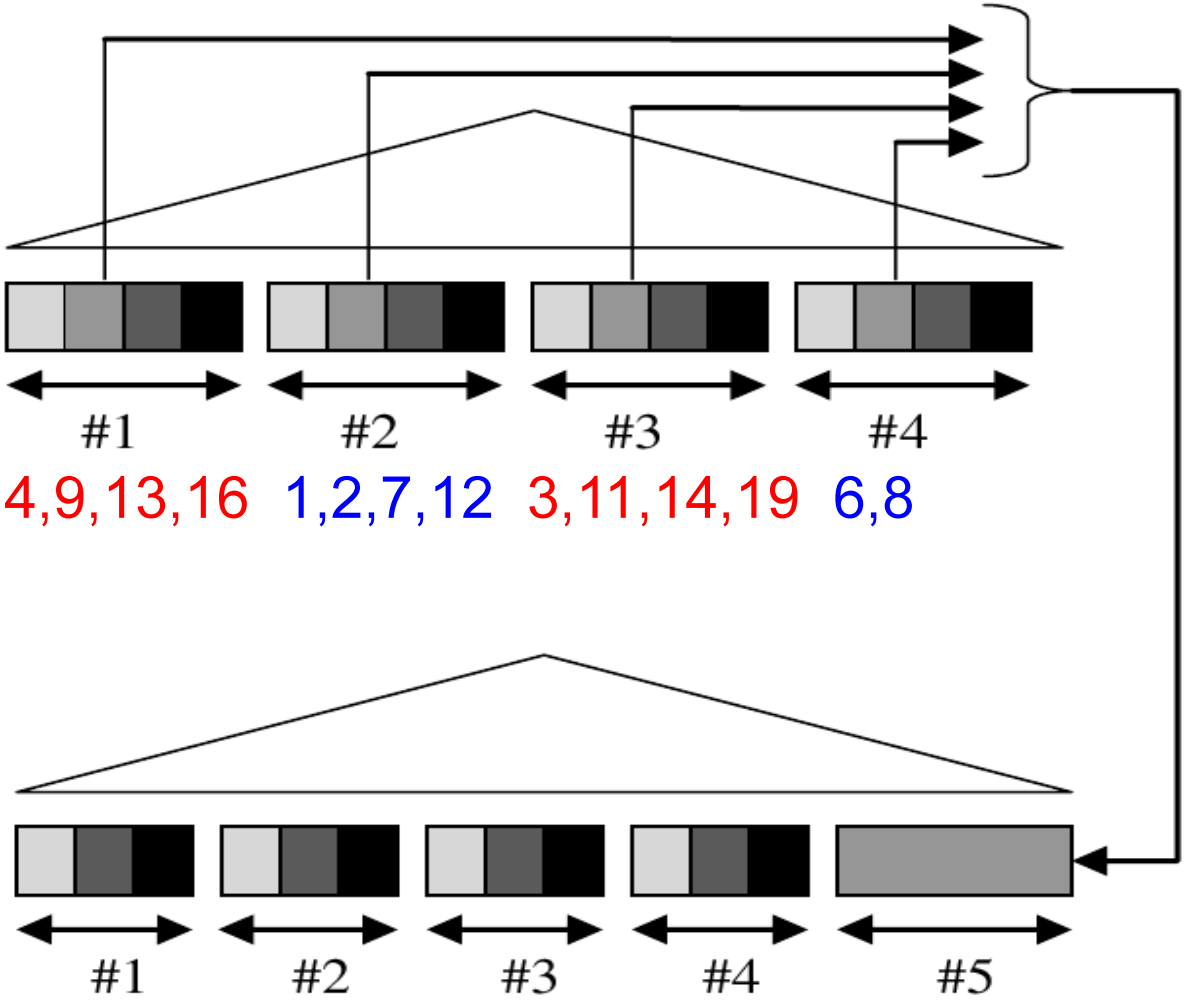


- Create partitioned B-tree using quicksort on cache-sized slices
- Partition have overlapping key ranges

13,16,4,9,2,12,7,1,19,3,14,11,8,6

Adaptive Merging

- Merge requested key range into single partition



Q1: 10 < R.A < 14: 11,12,13

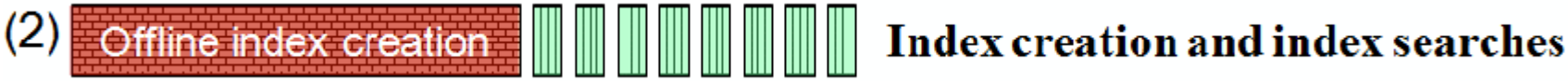
Q2: 7 < R.A <= 16: 8,9,11,12,13,14

4 1,2,7 3,19 6

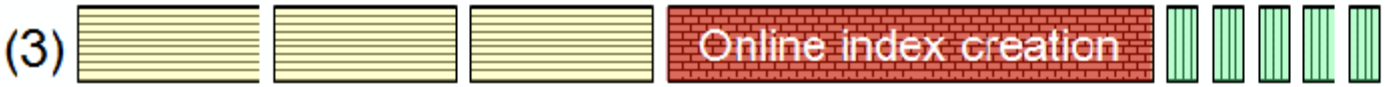
“Comparison”

- Database Cracking:
  - Designed for byte-addressable storage and consecutive arrays
  - Lower overhead (time) on first query
  - Slower convergence (number queries) to complete index
- Adaptive Merging:

Benchmarking: *Offline & Online Indexing*

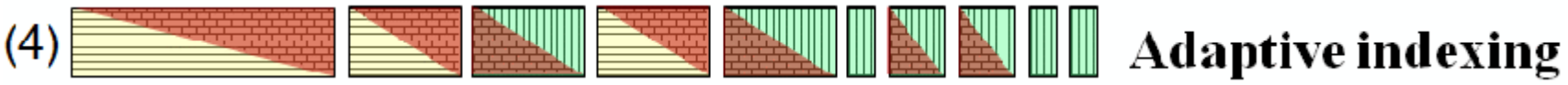


- Traditional benchmarks consider only query costs



- Schnaitter, Polyzotis; ICDE 2009:
  - Benchmark for online index creation
  - Considers also index creation costs
  - Distinct alternating phases
    - Query processing (incl. monitoring)
    - Index creation
  - Metrics:
    - Time (cost) to recognize promising indexes
    - Time (cost) to build indexes

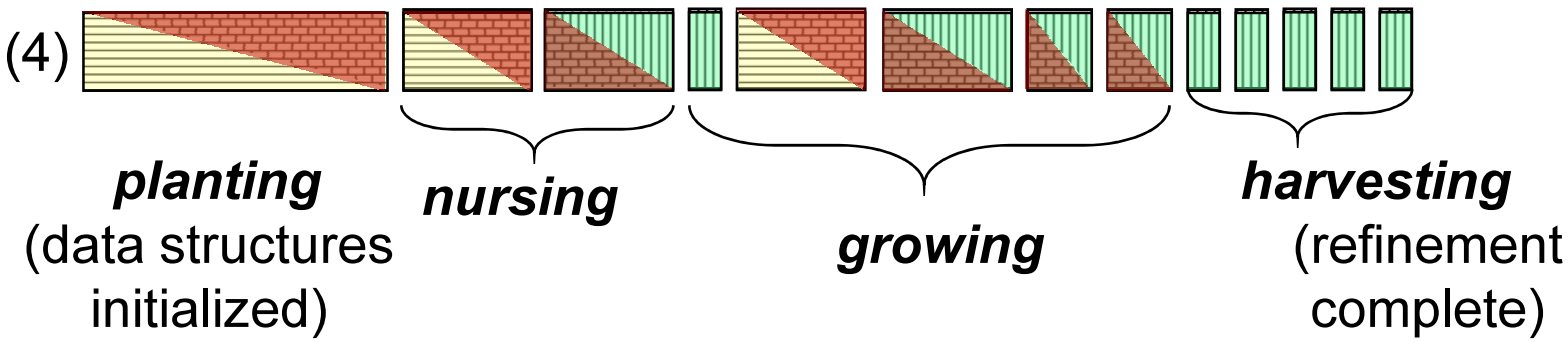
Benchmarking: *Adaptive Indexing*



- Indexes built continuously, not at once
  - No distinct index construction costs
  - No distinct phases (index construction / query processing)
  - No distinct query costs (without / with index)

- Incremental indexing adds overhead to each query
- Overhead changes over time
- Cluster amount of overhead to identify different stages

Adaptive Indexing: *Stages*



- ***Planting:***
  - Investments exceed benefits

  => Per-query costs higher than scan-based baseline
- ***Nursing:***
  - Investments start paying off

  => Per-query costs lower than scan-based baseline;
  Cumulative costs over all queries still higher than scan-based baseline
- ***Growing:***
  - Index structure starts converging to an optimal state

  => Also cumulative costs drop below scan-based baseline;
  First queries that do not require indexing side effects
- ***Harvesting:***
  - Index structure fully optimized

  => No more indexing required

Benchmarks: Workload Characteristics

- Dynamic environments:
  - Workload W: sequence of phases: $W = \{P_1, ..., P_n\}$
  - Phase P_i: sequence of queries and scheduling discipline: $P_i = (Q_i, S_i)$
- Stages occur per phase
  - First phase starts with planting stage
  - Subsequent phases may skip initial stages
    - Benefit from previous phases, e.g., due to partial overlap
  - A phase may not reach all stages
    - E.g., too short to reach optimal state

Benchmarks: Workload Parameters

- Range and distribution of keys within phases: focused vs. spread-out
- Length of phases
- Overlap between phases
- Query diversity per phase: number of columns & tables used
- Concurrency / scheduling

Benchmarks: Metrics

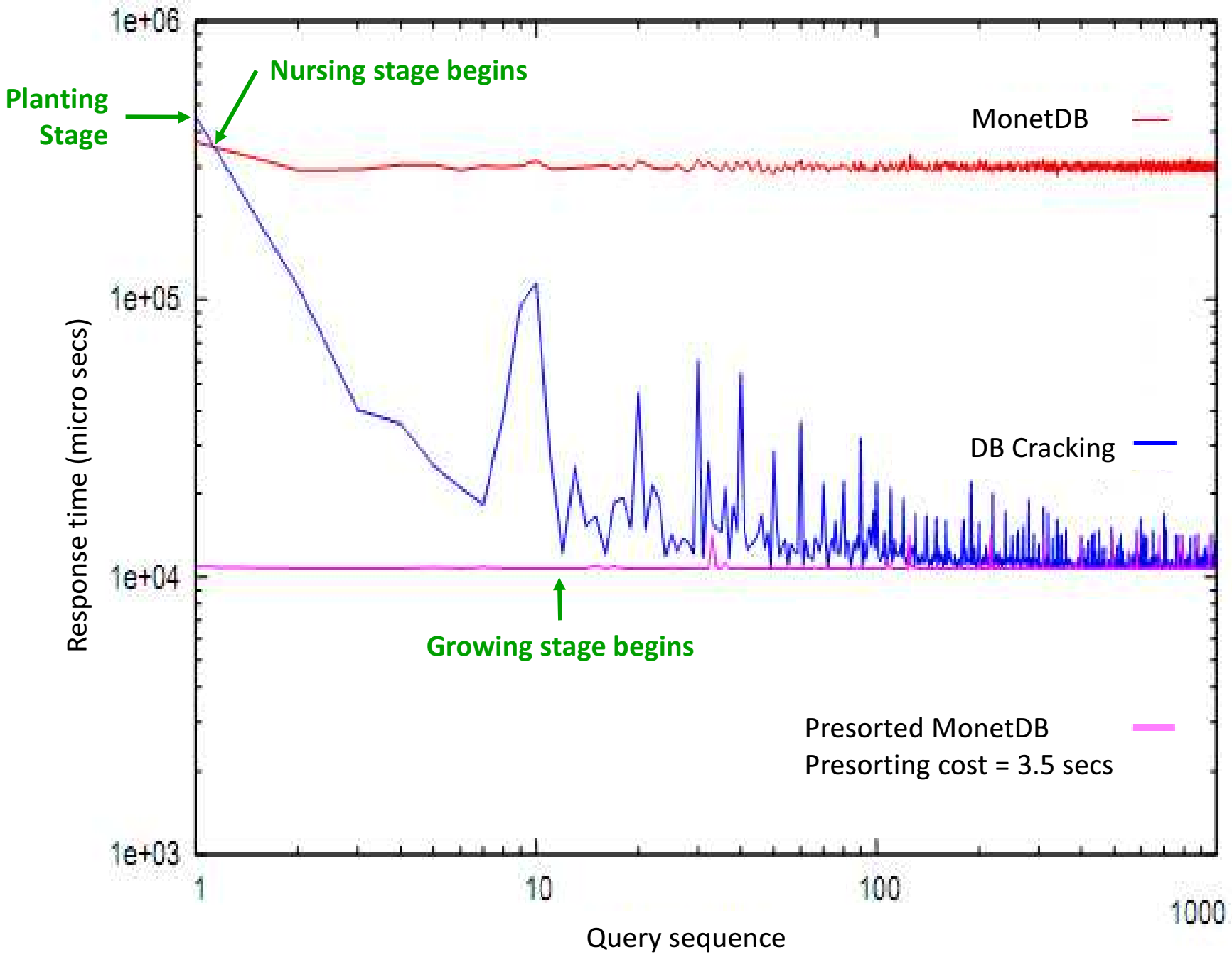
- Per-query & cumulative costs:
  - Time
  - Tuples accessed
  - Power consumption
  - ...
- Convergence / length of phases:

Sample Implementation and Experiments

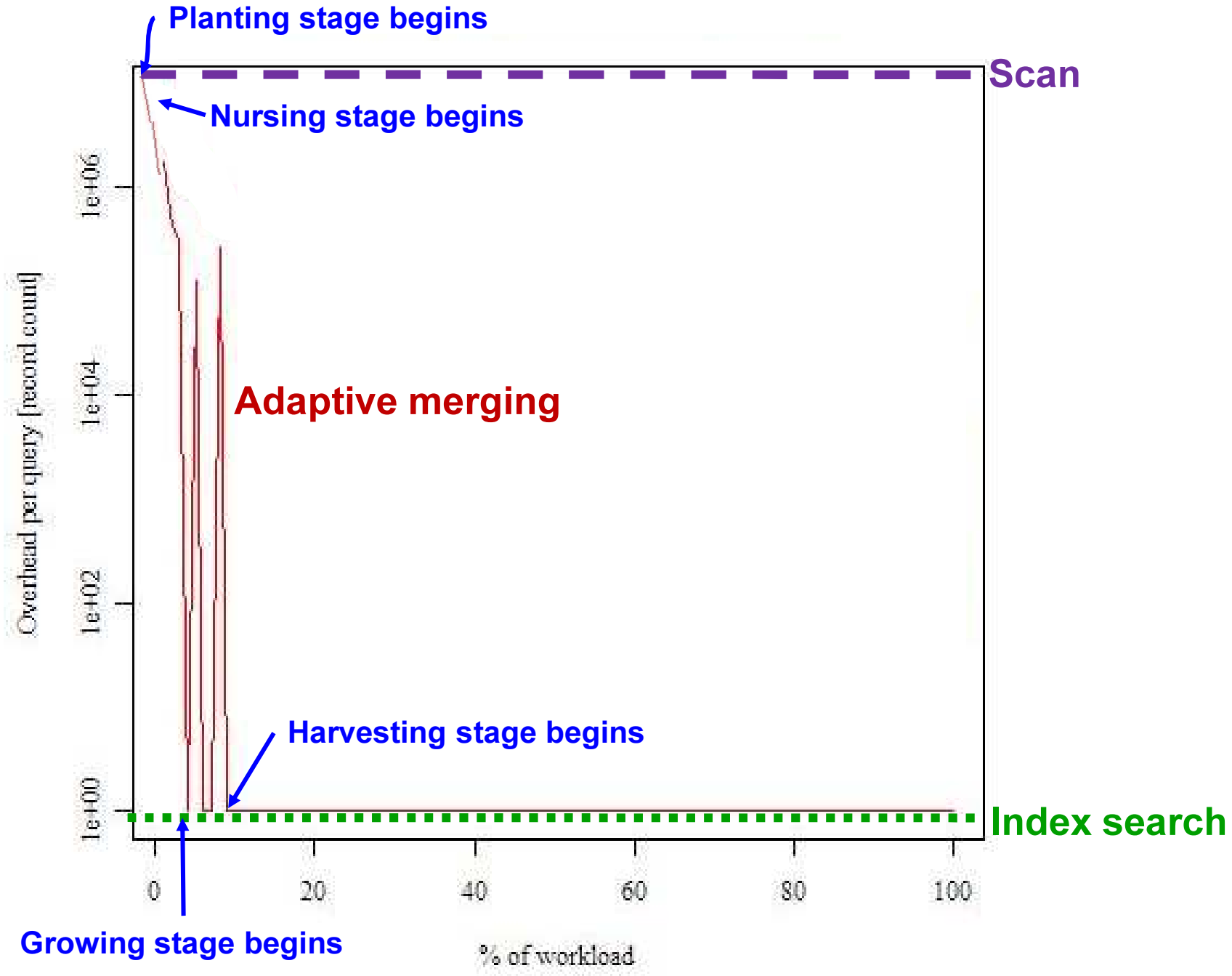
- Database Cracking:
  - Implemented in MonetDB (www.monetdb.org)
- Adaptive Merging:
  - Simulation experiments
  - (Implementation in MonetDB in progress)

- 1000 range selects over 10M tuples
  - 9/10 randomly in first half of domain
  - 1/10 randomly in second half of domain

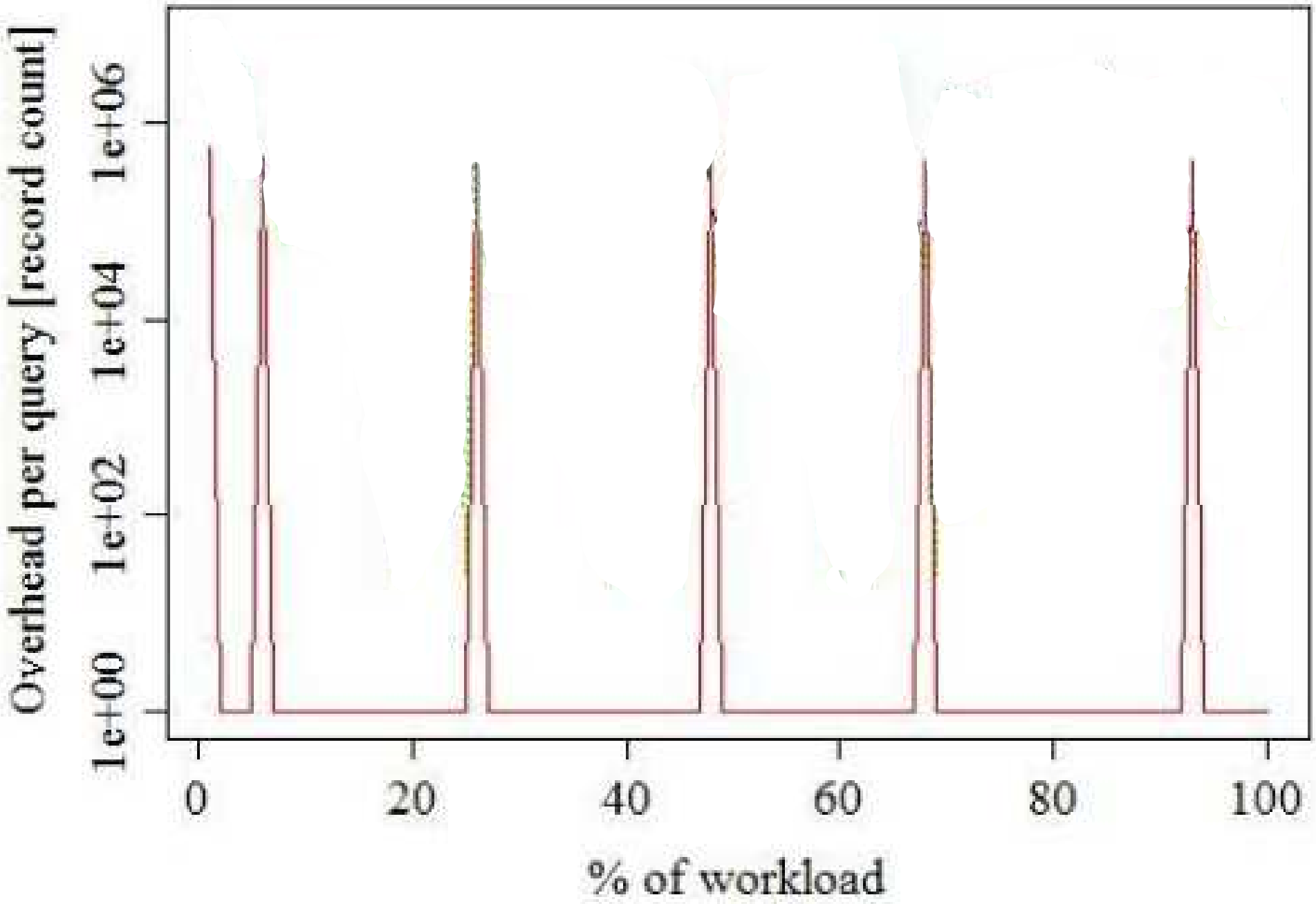
Adaptive Indexing Stages: *Database Cracking*



Adaptive Indexing Stages: *Adaptive Merging*



Adaptive Merging: Multiple Phases, shifting focus



Thank you!